Computer Memory

• Memory is used to store data and instructions in computers.

• There are different types of memory within a computer: registers, cache, main memory (primary memory), secondary memory (external memory). A computer may have all or a subset of these memory types.

• The different types of memories can be characterized by their speed, cost per bit and

capacity.

o Speed: How fast can data be accessed from the memory. This is defined by the memory access time (latency). Access time is the time from the instant that an address is presented to the memory to the instant that data have been stored or made available for use.

• The above characteristics for a certain memory type depend on the technology used to manufacture the memory. The technologies used to manufacture the memory types mentioned above and their characteristics is summarized below.

o Cache: Uses SRAM (Static Random Access memory)

• SRAM is made up of transistors (4 to 6 transistors per bit)

• Fast (Approximately 2 ns access time)

• Expensive ($5 per Megabyte)

o Main Memory: Uses DRAM (Dynamic Random Access Memory)

• DRAM is made up of transistors and capacitors (1 transistor and 1 capacitor per bit)

• Slower than SRAM (Approximately 60 ns access time)

• Less expensive than SRAM ($0.012 per Megabyte)

o Secondary Memory: Different types (Flash memory, magnetic disks like a hard disk, optical disks like a CD-ROM)

• These memory types are the slowest

• They are the least expensive

• We want to have fast memory with big capacity. But as you can see, as the speed for a certain memory type increases the price also increases. Having 100s of Gigabytes of the fastest memory only will be very expensive. Therefore a hierarchy of different types of

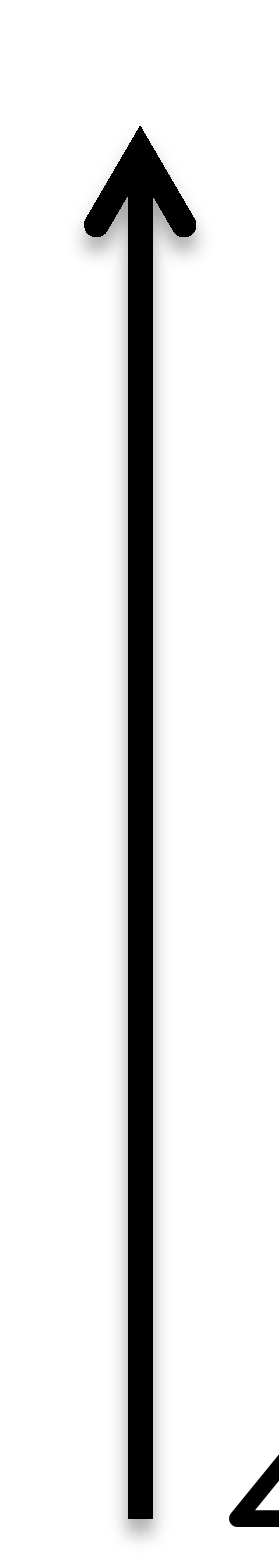
memory is used in computers.

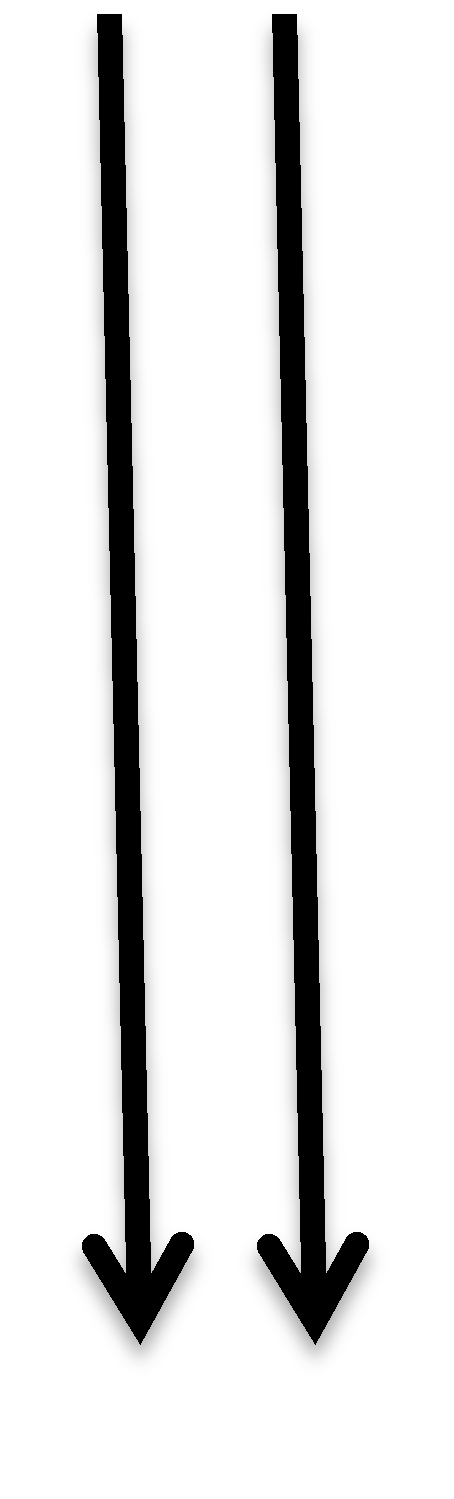




























Faster per bit

Registers

Cache

In Bytes

In Mbytes

Cheaper per bit

Increasing

Capacity

Main Memory

In Gbytes

Secondary Memory

In 100s of

Gbytes

• Use a small array of SRAM (cache), larger DRAM (main memory) and even larger secondary memory to fulfill the need for speed and capacity with a reasonable cost.

o Secondary memory permanently holds programs and data used by the computer (it

is non-volatile).

o Main memory holds instructions for current programs run by the computer (it is

volatile).

o Cache holds a copy of portion of main memory most recently accessed by the computer. Since, according to the principle of locality of reference, the most recently accessed memory location tend to be accessed again soon, keeping this data in faster memory (cache) decreases the average memory access time.

o The principle of locality of reference states that, if a data location is referenced, then the same location or data locations with nearby addresses will tend to be referenced soon. This arises from natural program structures. For example most programs contain loops, so instructions and data are likely to be accessed repeatedly.

Cache Memory

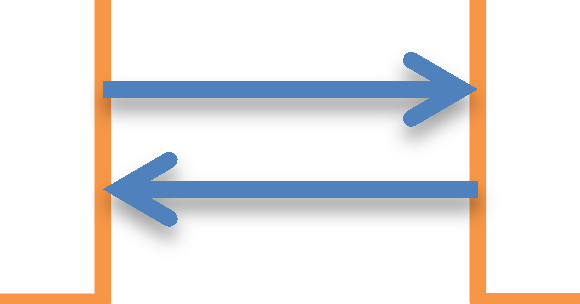
• A cache memory is logically located between a CPU and main memory (physically it is usually embedded inside the CPU). It contains a copy of portions of memory most recently accessed by the CPU.

• A processor may have a single cache or multiple levels of cache. Also there may be separate instruction and data cache (called split caches), or a single cache to hold both instruction

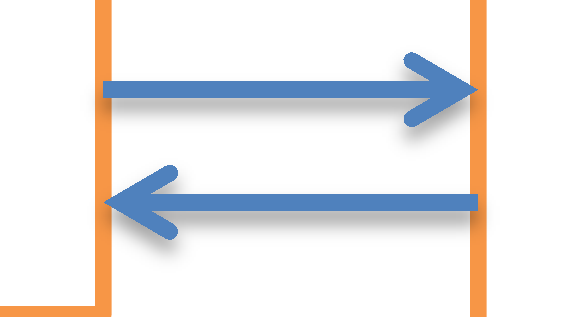
and data (called unified cache).











CPU Cache Main Memory

• When the CPU attempts to read a word from memory, a check is made to determine if the word is in cache. If so, the word is delivered to the processor (this is called a Hit). If the data is not in cache (this is called a Miss), a block of memory (several memory words)

consisting of that data is read into the cache and then the required word is delivered to the

CPU.

Cache structure

• Let the main memory of a computer contain 2n addressable words, with each word having a unique n-bit address. This memory can be divided into blocks, each block containing a number of addressable words.

• Let K = the number of words per block. This implies that there are 2n/K = M blocks in main memory as shown in the following diagram.

0









1 Block 0







2 (K Words)

2 n-1



Block M-1 (K Words)

• A cache memory consists of multiple tag/block pairs called cache lines. Let us assume a cache has L lines. The cache structure is shown below.

Control

Bits Tag Block (Data)

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

0

1

2

L-1

Block length

(K Words)



• Each cache line contains control bits, a tag field used in addressing, and a block of memory data.

• The number of cache lines is considerably less than the number of main memory blocks (L<<M). At any time, some subset of the blocks of memory resides in lines in the cache. If a word in a block of memory is read, that block is transferred to one of the lines of the cache. Because there are more blocks than lines, an individual line cannot be uniquely and permanently dedicated to a particular block. Thus, each line includes a tag that identifies which particular block is currently being stored. The tag is usually a portion of the main memory address.

• Memory (main memory) address is specified in instructions. A processor has to know

where in cache to look for a certain data, given the memory address. Therefore, the memory address specified in instructions has to be translated into cache line number. This translation of memory address into a cache line is termed as mapping. There are different mapping techniques: Direct Mapping, Associative Mapping and Set Associative Mapping.

a. Direct Mapping

• In direct mapping, each memory location (address) is mapped to exactly one location

(line) in cache.

• Let:

i = cache line number

j = main memory block number

L = number of lines in the cache

Then the cache line number for a certain memory block number is given as:

i = j % L

Example 1:

Assume main memory has 8 addressable words. Each memory block contains 1 memory word (that means there are 8 blocks of memory). Assume we have a direct mapped cache with two lines.

Control Tag Data

|  |  |  |
| --- | --- | --- |
|  | Main  Memory |  |
| 0 |  | Block 0 |
| 1 |  | Block 1 |
| 2 |  | Block 2 |
| 3 |  | Block 3 |
| 4 |  | Block 4 |
| 5 |  | Block 5 |
| 6 |  | Block 6 |
| 7 |  | Block 7 |

0

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

1

With this arrangement, blocks 0, 2, 4 and 6 can only be copied to cache line number 0. And blocks 1,

3, 5 and 7 can only be copied to cache line number 1.

• For direct mapping, memory address is mapped or interpreted as a tag, line number and block offset (word field).

Memory Address

Tag Line Number Block Offset

• For the above example we have a 3 bit memory address. Since a block contains only one word of memory, there is no need for block offset. Therefore, the two MSBs of the memory address are used as a tag, and the LSB of the memory address is used to identify cache line number.

Example 2:

What is the state of the direct mapped cache after executing the following sequence of instructions? (Assume a memory block size of 1 word.)

LDR R0, 2

LDR R1, 4

LDR R2, 5

V Tag Data

0

|  |  |  |
| --- | --- | --- |
| 0 |  |  |
| 0 |  |  |

1

R0

R1

R2

Main

Memory

0

|  |
| --- |
| 100 |
| 110 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

1

2

3

4

5

6

7

V bit in the cache indicates validity

of data in that line. V = 0 means the data available at that line is invalid.

Miss

V Tag Data

0

|  |  |  |
| --- | --- | --- |
| 1 | 01 | 120 |
| 0 |  |  |

1

R0 120

R1

R2

LDR R0, 2 (010)

Main

Memory

|  |
| --- |
| 100 |
| 110 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

0

1

2

3

4

5

6

7

Miss

V Tag Data

|  |  |  |
| --- | --- | --- |
| 1 | 10 | 140 |
| 0 |  |  |

0

1

R0 120

R1 140

R2

LDR R1, 4 (100)

Main

Memory

|  |
| --- |
| 100 |
| 110 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

0

1

2

3

4

5

6

7

LDR R2, 5 (101)

Miss

V Tag Data

0

|  |  |  |
| --- | --- | --- |
| 1 | 10 | 140 |
| 1 | 10 | 150 |

1

R0 120

R1 140

R2 150

Main

Memory

|  |
| --- |
| 100 |
| 110 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

0

1

2

3

4

5

6

7

Example 3:

Repeat example 2 for a memory block size of 2 words. (In this case there are two words per cache line; therefore one bit of memory address is used as a block offset.)

V Tag Data

|  |  |  |
| --- | --- | --- |
| 0 |  |  |
|  | |  |
| 0 |  |  |
|  | |  |

0

1

Main

Memory

0

|  |
| --- |
| 100 |
| 110 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

1

2

3

Block 0

Block 1

3 bits

Tag Line

No.

Block

Offset

R0 4

R1 5

R2 6

7

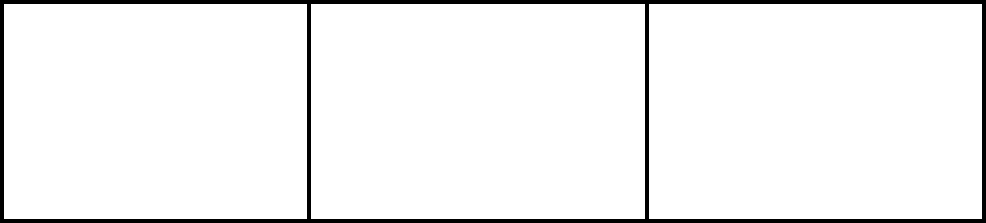
Block 2

Block 3

1 1 1

Memory Address

LDR R0, 2 (010)



Miss

V Tag Data

0

|  |  |  |
| --- | --- | --- |
| 0 |  |  |
|  | |  |
| 1 | 0 | 120 |
|  | | 130 |

1

R0 120

R1

R2

Main

Memory

|  |
| --- |
| 100 |
| 110 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

0

1

2

3

4

5

6

7

Block 0

Block 1

Block 2

Block 3

Miss

V Tag Data

0

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 140 |
|  | | 150 |
| 1 | 0 | 120 |
|  | | 130 |

1

R0 120

R1 140

R2

LDR R1, 4 (100)

Main

Memory

|  |
| --- |
| 100 |
| 110 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

0

1

2

3

4

5

6

7

Block 0

Block 1

Block 2

Block 3

LDR R1, 5 (101)

Hit

V Tag Data

0

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 140 |
|  | | 150 |
| 1 | 1 | 120 |
|  | | 130 |

1

R0 120

R1 140

R2 150

• Hit ratio increases as the block size increases, due to locality of reference.

• The direct mapping technique is simple and inexpensive to implement. Its main disadvantage is that there is a fixed cache location for any given block. Thus, if a program happens to reference words repeatedly from two different blocks that map into the same line (for example block 0 and block 2 in the above example), then the blocks will be continually swapped in the cache, and the hit ratio will be low (a phenomenon known as

thrashing).

Example 4:

How many tag bits are required for :

• 32-bit memory address

• Direct mapped 32 Kbytes of cache

• 128 byte block size

Number of cache lines = 32K / 128 = 256 => 8 bits for line number

128 bytes / block => 7 bits for block offset

Tag bits = memory address bits – line number bits – block offset bits

Tag bits = 32 – 8 – 7 = 17 bits

Note: Cache size only specifies the amount of data that can be stored in the cache; it doesn’t include bits used for tag and other control bits

Note: 32K is actually 215 =

32768 (not 32000)

b. Associative (Fully Associative) Mapping

• Permits each main memory block to be loaded (copied) into any line of the cache.

• For this type of mapping memory address is mapped or interpreted as a tag and block offset (word field). The cache line number does not give any information, since memory

address can be mapped to any line number.

Memory Address

Tag Block Offset

• To determine whether a block is in the cache, a cache control logic must simultaneously examine every line’s tag for a match (need extra circuit that does this).

Example 5:

Consider a main memory with 32 blocks. In fully associative mapping the tag field uniquely identifies a block of main memory; therefore a 5 bit tag is needed.

Example 6:

What is the state of the fully associative mapped cache after executing the following sequence of instructions? (Assume a memory block size of 2 word and the number of cache lines are 2.)

V Tag Data

|  |  |  |
| --- | --- | --- |
| 0 |  |  |
|  | |  |
| 0 |  |  |
|  | |  |

R0

R1

R2

LDR R0, 2

LDR R1, 3

LDR R2, 5

Main

Memory

0

|  |
| --- |
| 100 |
| 110 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

1

2

3

4

5

6

7

Block 0

Block 1

Block 2

Block 3

LDR R0, 2 (010)

There are four blocks;

therefore a 2 bit tag is required.

Miss

V Tag Data

|  |  |  |
| --- | --- | --- |
| 1 | 01 | 120 |
|  | | 130 |
| 0 |  |  |
|  | |  |

R0 120

R1

R2

Main

Memory

|  |
| --- |
| 100 |
| 110 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

0

1

2

3

4

5

6

7

Block 0

Block 1

Block 2

Block 3

The two MSBs of a memory address correspond to the tag field.

The LSB of a memory address correspond to block offset.

LDR R1, 3 (011)

V Tag Data

Hit

|  |  |  |
| --- | --- | --- |
| 1 | 01 | 120 |
|  | | 130 |
| 0 |  |  |
|  | |  |

R0 120

R1 130

R2

LDR R2, 5 (101)

Miss

V Tag Data

|  |  |  |
| --- | --- | --- |
| 1 | 01 | 120 |
|  | | 130 |
| 1 | 10 | 140 |
|  | | 150 |

Main

Memory

0

|  |
| --- |
| 100 |
| 110 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

1

2

3

4

5

6

7

Block 0

Block 1

Block 2

Block 3

• Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced. For direct mapping, there is only one possible line for any particular block, and no choice is possible. For the associative technique (and set- associative technique we will discuss later), however, a replacement policy (algorithm) is needed.

|  |  |
| --- | --- |
| R0 | 120 |
| R1 | 130 |
| R2 | 150 |

• A number of algorithms are available:

o Least Recently Used (LRU): Replace the block that has been in the cache longest with no reference to it. The cache mechanism maintains a separate list of indexes to all the lines in the cache. When a line is referenced, it moves to the front of the list. For replacement, the line at the back of the list is used. Because of its simplicity of implementation, LRU is the most popular replacement algorithm.

o First-In-First-Out (FIFO): Replace the block that has been in the cache longest. This can be easily implemented by replacing blocks in round-robin fashion.

o Least Frequently Used (LFU): Replace that block in the set that has experienced the fewest references. With this approach, there must be a counter associated with each line that records the number of references to that line.

o Pick Randomly: pick a line at random from among the lines.

c. Set-Associative Mapping

• This type of mapping is a compromise that exhibits the strengths of both the direct and fully associative approaches while reducing their drawbacks.

• In this type of mapping each main memory block can only be loaded (copied) into a set of lines (more than one line) of the cache.

• For set-associative mapping, the cache control logic interprets a memory address as three

fields: Tag, Set Index, and Block Offset (Word).

Memory Address

Tag Set Index Block Offset

Example 7:

Consider a main memory with 16 addressable words. Each memory block contains two memory words (there are 8 blocks). Assume the system has a set-associative cache with 4 lines. Let us divide the cache into two sets, each set containing two lines.

Tag Data

Set 0

Set 1

Main

Memory

0

1

2

3

4

5

6

7

8

9

Block 0

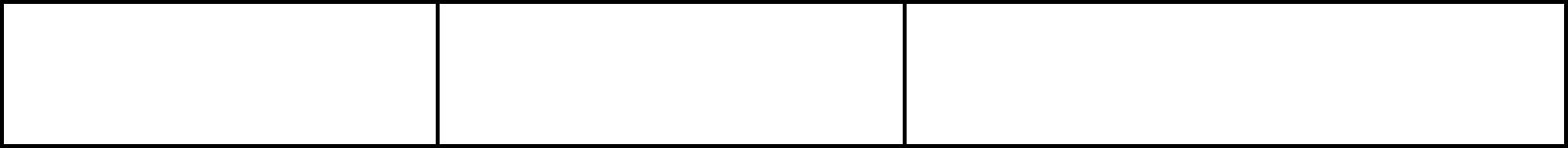
Block 1

Block 2

Block 3

Block 4

Main memory address is 4 bits: 10



11 o Out of this 1 bit is used to identify one 12 of the two sets 13

o 1 bit is used as block offset 14

o 2 bits are used as a tag 15

Tag Set Index Block Offset

2 bit 1 bit 1 bit

Block 5

Block 6

Block 7

• Using this setup, Blocks 0, 2, 4 and 6 can only be copied to one of the two cache lines in set

0. Blocks 1, 3, 5 and 7 can only be copied to one of the two cache lines in set 1.

• In this example there are two lines per set. Such mapping is referred to as two-way set associative mapping.

Example 8:

What is the state of the set-associative mapped cache given in example 7, after executing the following sequence of instructions?

LDR R0, 2

LDR R1, 15

0

0

0

0

Main

Memory

|  |
| --- |
| 100 |
| 110 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |
| … |
| 0 |
| 0 |

1

3

5

7

14

15 Block 7

V Tag

LDR R0, 2 (0010)

Tag = 00

Set index = 1

block offset = 0

2

|  |  |  |
| --- | --- | --- |
| 0 |  |  |
|  | |  |
| 0 |  |  |
|  | |  |
| 1 | 00 | 120 |
|  | | 130 |
| 0 |  |  |
|  | |  |

|  |
| --- |
| … |
| 120 |
| 130 |
| … |

3

Miss

R0 120

R1

R2

LDR R1, 15(1111)

Miss

V Tag Data

…

14 0

15 0

R0 120

|  |  |  |
| --- | --- | --- |
| 0 |  |  |
|  | |  |
| 0 |  |  |
|  | |  |
| 1 | 00 | 120 |
|  | | 130 |
| 0 | 11 | 0 |
|  | | 0 |

R1 0

R2

• In set-associative mapping, one of the previously mentioned replacement policies must be used when all the lines in a set have been filled.

Write Policy

• So far we have been considering memory read operations only. But there are also instructions that write to memory (instruction that modify the content of memory). If at least one write operation has been performed on a word in a line of cache, then main memory must be updated by writing the line of cache out to the block of memory if that line needs to be replaced.

• There are a variety of write policies (techniques). Two of these are write through and

write back.

• When using write through technique, all write operations are made to main memory as well as to the cache, ensuring that main memory is always valid.

Example 9:

Consider a direct mapped cache that uses write through technique. What will be the state of the cache after execution of the following instructions?

STR R0, 1

LDR R1, 4

V Tag Data

0

|  |  |  |
| --- | --- | --- |
| 0 |  |  |
|  | |  |
| 0 |  |  |
|  | |  |

1

R0 120

R1 0

R2 0

Main

Memory

0

|  |
| --- |
| 100 |
| 110 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

1

2

3

4

5

6

7

Block 0

Block 1

Block 2

Block 3

STR R0, 1(001)

Miss

V Tag Data

0 1 0 100

120

1 0

Main

Memory

|  |
| --- |
| 100 |
| 120 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

0

1

2

3

4

5

6

7

Block 0

Block 1

Block 2

Block 3

LDR R1, 4(100)

|  |  |  |
| --- | --- | --- |
|  | R0 | 120 |
| R1 | 0 |
|  | R2 | 0 |

Miss

V Tag Data

0

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 140 |
|  | | 150 |
| 0 |  |  |
|  | |  |

1

R0 120

R1 140

R2 0

Main

Memory

0

|  |
| --- |
| 100 |
| 120 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

1

2

3

4

5

6

7

Block 0

Block 1

Block 2

Block 3

• Another writing technique is write back. With this technique, updates are made only in the cache. When an update occurs, a dirty bit, or use bit, associated with the line is set. Then, when a block is replaced, it is written back to main memory if and only if the dirty bit is set.

• Write back technique minimizes memory writes compared to write through.

Example 10:

Consider the previous example. If a write back technique is used, execution of the instructions in example 9 would look like the following. Note that a dirty bit (D) is added in the cache as a control

bit.

Miss

V D Tag Data

0

1

R0 120

R1 0

R2 0

STR R0, 1(001)

LDR R1, 4(100)

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 1 | 0 | 100 |
|  | | | 120 |
| 0 |  |  |  |
|  | | |  |

Main

Memory

|  |
| --- |
| 100 |
| 110 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

0

1

2

3

4

5

6

7

Block 0

Block 1

Block 2

Block 3

Miss

V D Tag Data

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 1 | 0 | 100 |
|  | | | 120 |
| 0 |  |  |  |
|  | | |  |

0

1

R0 120

R1 0

R2 0

Main

Memory

|  |
| --- |
| 100 |
| 120 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

0

1

2

3

4

5

6

7

Block 0

Block 1

Block 2

Block 3

V D Tag Data

0

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 0 | 1 | 140 |
|  | | | 150 |
| 0 |  |  |  |
|  | | |  |

1

R0 120

R1 140

R2 0

Main

Memory

0

|  |
| --- |
| 100 |
| 120 |
| 120 |
| 130 |
| 140 |
| 150 |
| 0 |
| 0 |

1

2

3

4

5

6

7

Block 0

Block 1

Block 2

Block 3

References

1. Computer Architecture and Organization, William Stallings, 8th edition

2. Computer Organization and Design, David A. Patterson, 4th edition